VISIT…

OPTIMISATION OF ANALOG CIRCUITS WITH *SPICE* 3F4

Janez Puhan Tadej Tuma

University of Ljubljana, Faculty of Electrical Engineering, Tr aška 25, 1000 Ljubljana, Slovenia,
janez.puhan@fe.uni-lj.si, tadej.tuma@fe.uni-lj.si

***Abstract* - In the paper four approaches of upgrading analog circuits analysis tools (*SPICE* was used) with optimisation loops are compared. In the first approach the optimisation algorithm is coded in *Mathematica*, which calls *SPICE* for every circuit analysis. In the second approach the algorithm is coded in *Nutmeg*. (*Nutmeg* is an interpreter designed as a user interface for *SPICE*, but it has all properties of a programming language.) In the third approach the algorithm is again coded in *Nutmeg*, but all computation (for example computation of the cost function) is performed by extern programs. And finally in the fourth approach the optimisation algorithm is coded in *C*. We compared computation times of all different approaches with a constrained direct optimisation method on a simple circuit. It turned out that the approach through *Mathematica* is elegant, but hopelessly slow. Both approaches through *Nutmeg* are compact, but unfortunately *Nutmeg* yields numerous bugs. The best results were obtained with *C* programs.**

I. INTRODUCTION

Digital computers have been successfully used in circuit design for more decades. Analysis and simulation of different kinds of analog and digital circuits can be done. The most popular software package for computer aided design in this moment is *SPICE*.

Optimisation methods are the next step in automatic design of analog circuits. They require a given circuit topology and partially circuit parameters. Some parameters are left undetermined. The user gives his »wishes« about some circuit properties in a form of a cost function instead of determining parameters. The optimisation method then calculates the most appropriate values for the undetermined parameters [1].

SPICE is very reliable and has a lot of users around the world, but the latest version (we used Berkeley's *SPICE* version 3f4) still does not offer optimisation.

There arises the idea about upgrading *SPICE* with optimisation methods. The main problem of optimisation is computing time, because a lot of circuit analyses have to be done. We tried to upgrade *SPICE* with optimisation loops in four different ways. The approaches using *Mathematica*, *Nutmeg*, *Nutmeg* with external programs and *C* program were tested. We were most interested in computing time.

II. DECSRIPTION OF DIFFERENT APPROACHES

As it was mentioned in the introduction we tried to upgrade *SPICE* in four different ways. We looked for the tool, which would allow easy description of the cost function and would be programmable at the same time. More or less difficult mathematical relations are usually present in the cost function. *Mathematica* [2] is on the other hand very appropriate for description of such mathematical functions. It also has its own interpreter programming language. So the optimisation loop was coded directly in *Mathematica*. For every circuit analysis *SPICE* was called separately. The communication between *Mathematica* and *SPICE* was established by two

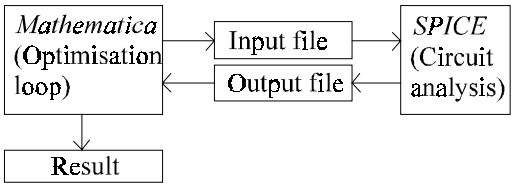

Figure 1: Approach with *Mathematica*

files. In the input file for *SPICE Mathematica* prepares data for the next circuit analysis. *SPICE* gives results in output file, which is then read by *Mathematica*. The situation is illustrated in figure 1.

In the second approach the algorithm was coded in *Nutmeg* [3]. *Nutmeg* is an interpreter language, which was designed as an interactive user interface for *SPICE*. It has all the properties of a real programming language

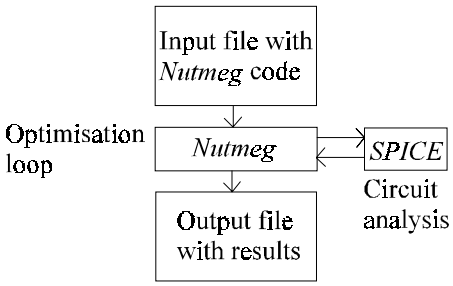

Figure 2: Approach with *Nutmeg*

and should therefore be convenient for coding optimisation loops. The communication through files would also be avoided. This solution is very compact and is shown in figure 2.

Unfortunately *Nutmeg* has serious problems with memory allocation. As a consequence of this problem, *Nutmeg* yields numerous bugs. So it is not capable to process a lot of variables. This is the reason why we tried to unload *Nutmeg*. All computation (for example computation of cost function ...) is performed by extern programs instead of *Nutmeg*. But communication through files is again introduced between *SPICE* and the extern programs. This approach is illustrated in figure 3.

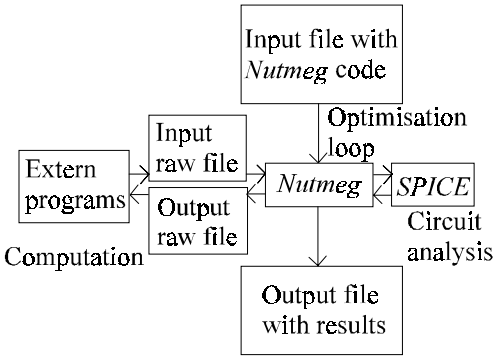

Figure 3: *Nutmeg* with extern programs

Nutmeg was free from the main part of computing in figure 3. But it was still not enough and the problem with memory allocation continued. So we finally tried with our own program. The source code is written in *C* language.

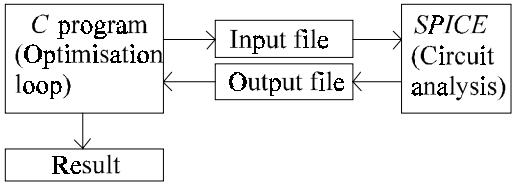

Figure 4: Approach with *C* program

The whole concept is similar to the first approach with *Mathematica*. Again *SPICE* is called for circuit analysis. The data flow between *SPICE* and *C* program again goes through input and output files (figure 4).

III. OUR SIMPLE BENCHMARK CASE

All four approaches were tested on a simple circuit. A transistor amplifier with a bipolar *npn* transistor in orientation with common emitter was used. The circuit is shown in figure 5. We wanted to amplify the input current. Its amplitude does not exceed 100µA

$$|i_1| \le I_M = 100\mu\text{A} \qquad \textbf{(1)}$$

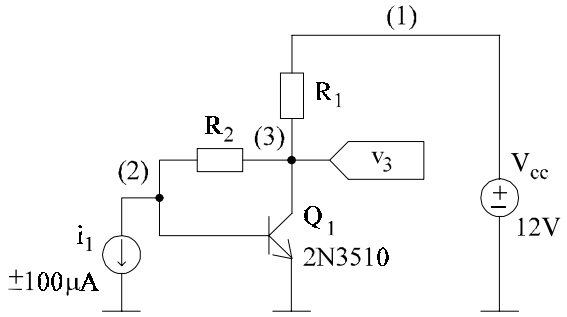

Figure 5: Transistor amplifier

The amplifier is supplied with direct voltage V_{cc} = 12V and is built around a standard bipolar transistor 2N3510. The resistors R_1 and R_2 determine the operating point and the feedback factor of the amplifier. They also have an influence on all other properties of the amplifier (input and output impedance, gain, degree of non-linearity, upper frequency limit, noise spectrum etc.). The output voltage is in general given by equation (2). We are

$$v_3 = v_3(i_1, R_1, R_2) \qquad \textbf{(2)}$$

interested in non-linear distortion of the output. We can estimate, that all other properties will be in acceptable boundaries, if the resistances R_1 and R_2 are inside these intervals

$$5\text{k}\Omega \le R_1 \le 50\text{k}\Omega$$
$$20\text{k}\Omega \le R_2 \le 200\text{k}\Omega \qquad \textbf{(3)}.$$

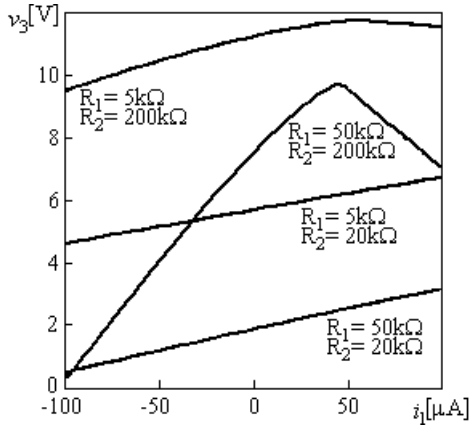

Figure 6: Direct transfer functions

Equation (3) defines explicit constrains. These constrains are often defined by technology. So, in this example, we search for the couple R_1 and R_2 inside these intervals, which give the most linear response and a gain factor defined as

$$A = \frac{v_3(I_M, R_1, R_2) - v_3(-I_M, R_1, R_2)}{2I_M} \qquad \textbf{(4)}$$

must satisfy the implicit constrain

$$|A| \geq 20\,\mathrm{kV/A} \qquad \textbf{(5)}$$

at the same time. The gain and non-linear distortion will be studied on a direct transfer function, which is also competent for low frequencies. Circuit analyses in all four corners of the parameter space were done for illustration and are shown in figure 6.

A unique criterion, which values of R_1 and R_2 are the best, is needed in the first place. In our case the rate of non-linearity have to be defined. The cost function is defined as

$$E(R_1, R_2) = \frac{\int_{-I_M}^{I_M} (Ai_1 - v_3(i_1) + v_3(0))^2 \,\mathrm{d}i_1}{\int_{-I_M}^{I_M} (Ai_1)^2 \,\mathrm{d}i_1} \qquad \textbf{(6)}.$$

Optimisation methods should find the minimum of a cost function with respect to explicit and implicit constrains. The constrained simplex optimisation method [4] was chosen. It is one of the most robust direct optimisation methods and is fairly simple from the programmers point of view. So this method was coded in all four approaches.

In our case the parameter space is two dimensional, so the cost function can be plotted. In figure 7 the cost

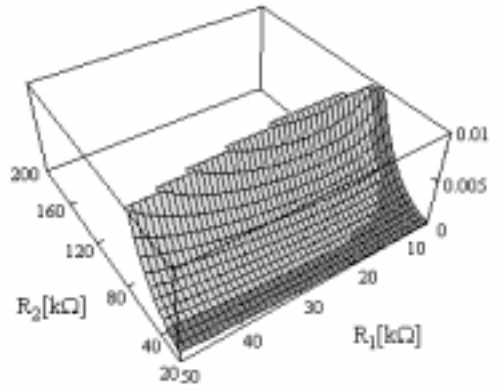

Figure 7: Cost function

function $E(R_1, R_2)$ is shown as a surface over explicitly constrained parameter plain. Transfer functions in all four corners were shown in figure 6. The surface is shaded where implicit constrains (equation (5)) are violated.

IV. RESULTS

The simple circuit introduced in previous paragraph was optimised in our four ways. The initial simplex was always the same (R_1 = 50kΩ R_2 = 200kΩ; R_1 = 40kΩ R_2 = 200kΩ; R_1 = 45kΩ R_2 = 170kΩ; R_1 = 50kΩ R_2 = 150kΩ) and the optimal values for the resistances R_1 and R_2 were sooner or later always found ($R_{1opt.} \cong 25.8$kΩ $R_{2opt.} \cong 33.85$kΩ (figure 7)). To obtain this optimal values approximately 107 circuit analyses were needed. All four approaches were tested on a Hewlett-Packard Apollo Series 400 Model 425e computer. Computation time was of our main interest. Figure 8 shows computing time needed by different approaches. As we can see from figure 8 *Nutmeg* is the fastest. The percentage of time

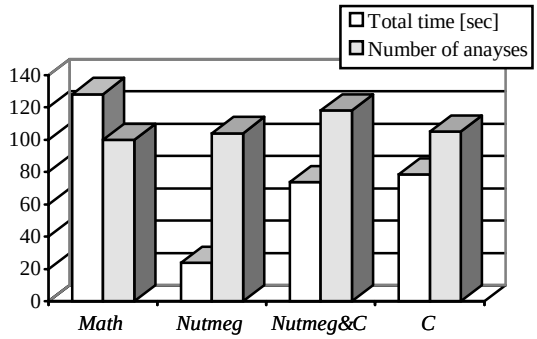

Figure 8: Computing time

Mathematica	1.282 seconds / iteration
Nutmeg	0.230 seconds / iteration
Nutmeg & *C*	0.625 seconds / iteration
C	0.746 seconds / iteration

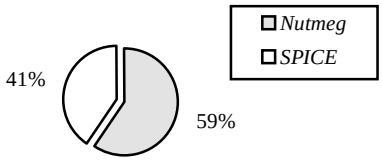

Figure 9: Distribution of computing time for *Nutmeg*

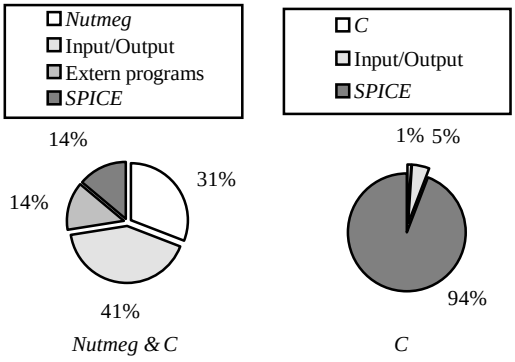

Figure 10: Distribution of computing time for *Nutmeg* & *C* and *C*

needed for circuit analyses and *Nutmeg* overhead is shown in figure 9. *Nutmeg* & *C* and *C* approaches are both much slower than *Nutmeg*, and they have approximately the same total time. Distribution of computing time for those two approaches is shown in figure 10. *Mathematica* is the slowest approach, so it would be interesting to see where *Mathematica* loses time. In figure 11 distribution of computing time for *Mathematica* is shown.

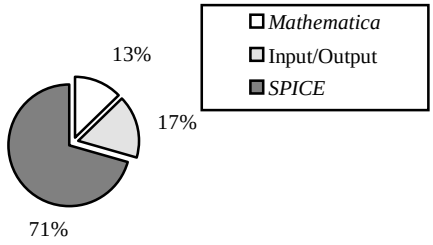

Figure 11: Distribution of computing time for *Mathematica*

We can see, that *Nutmeg* approach spends more than half of computing time for itself. *Nutmeg* is not a true programming language and that is the reason for such waste of time. The same problem occurs with the *Nutmeg* & *C* approach. Extern programs and circuit analyses both take less than one third of the computing time. Again *Nutmeg* spends a lot of time for itself, and a lot of time is spent for reading and writing input and output files. *C* and *Mathematica* approaches on the other hand spend the major of computing time for circuit analyses. *Mathematica* is slower, because it is an interpreter. In those two approaches *SPICE* is called for every circuit analysis separately. The consequence of this is longer time needed for one circuit analysis. This is shown in figure 12, where times needed for 100 circuit analyses in both cases (separate and non-separate calls) are compared.

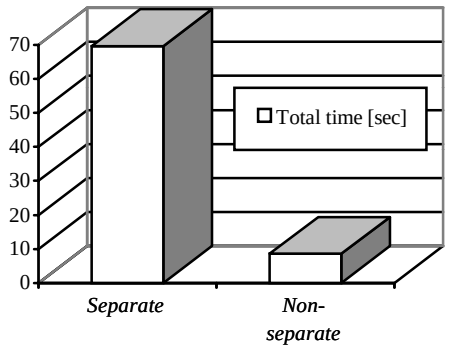

Figure 12: Time needed for 100 circuit analyses

V. CONCLUSIONS

It turned out, that *Nutmeg* has serious problems with memory allocation. For example a simple loop

```
let n = 1
dowhile n > 0
end
```

ends after finite number of steps. The reason is uncontrolled memory allocation, which results in lack of memory and memory error at the end. So bigger circuits can not be optimised with *Nutmeg*. Even the simple example in this paper was hard to optimise because of memory errors. The second drawback of *Nutmeg* is its interpreter nature. It can be expected, that in bigger circuit *Nutmeg* would spend more time for itself and less for circuit analyses. So it would become relatively slower. Exactly the same problems occur in *Nutmeg* & *C* approach. The approach through *Mathematica* is very elegant. The cost function can be described »user friendly«. However *Mathematica* is again an interpreter and therefore slower than *C*. Both, *C* and *Mathematica*, also call *SPICE* separately for every circuit analysis, which results in significant time loss. That is the main drawback of the *C* approach, and partly *Mathematica* approach as well.

For effective circuit optimisation *Nutmeg* would have to be improved, maybe even rewritten. Writing new optimisation tools in *C*, which would use *SPICE* for circuit analysis, would be maybe even more appropriate. *SPICE* would have to become a »daemon«, which would analyse circuit when needed. Loading and closing *SPICE* for every circuit analysis would be avoided this way.

References

[1] R. K. Brayton, R. Spence, *Sensitivity and Optimisation*, Elsevier Scientific Publishing Company, Amsterdam, 1980

[2] S. Wolfram, *Mathematica, A System for doing Mathematics by Computer*, Addison-Wesley Publishing Company, Redwood City, California, 1991

[3] T. Quarles, A. R. Newton, D. O. Pederson, A. Sangiovanni-Vincentelli, *SPICE3 Version 3f4 User's Manual*, University of California, Berkeley, California, 1989

[4] M. J. Box, *A new method of constrained optimisation and comparison with other methods*, Computer Journal, vol. 7, pages 42-52, 1965